

Memory Technologies

- **Random access technologies:**
 - “Random” good: access time same for all locations
 - DRAM: Dynamic Random Access Memory
 - High density, low power, cheap, but slow
 - Dynamic: need to be “refreshed” regularly
 - SRAM: Static Random Access Memory
 - Low density, high power, expensive, fast (2–10x DRAM)
 - Static: content will last “forever”
- **“Not-so-random” access technologies:**
 - Access time varies by location at any time
 - Typically, 10^5 – 10^8 times slower than DRAM
 - Examples: Disk, tape drive, CDROM
- **Multiple levels of memory hierarchy:**
 - among random access technologies: SRAM/DRAM
 - random to non-random: DRAM to disk

Memory, 5

Technology Trends

	<u>Capacity</u>	<u>Speed</u>
Logic:	2x in 3 years	2x in 3 years
DRAM:	4x in 3 years	1.4x in 10 years
Disk:	2x in 3 years	1.4x in 10 years

DRAM		
<u>Year</u>	<u>Size</u>	<u>Cycle Time</u>
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns

Memory, 6

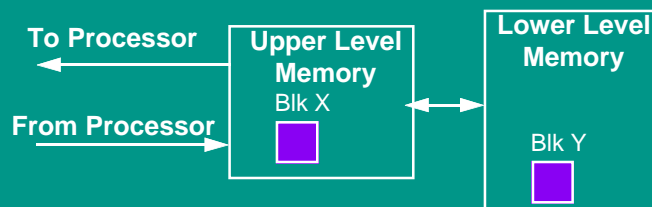
The Principle of Locality

- **The Principle of Locality:**
 - Program accesses a relatively small portion of the address space at any instant of time.
- **Two Different Types of Locality:**
 - **Temporal Locality (Locality in Time):** If an item is referenced, it will tend to be referenced again soon.
 - **Spatial Locality (Locality in Space):** If an item is referenced, items whose addresses are close by tend to be referenced soon.

Memory, 7

Memory Hierarchy: Principles

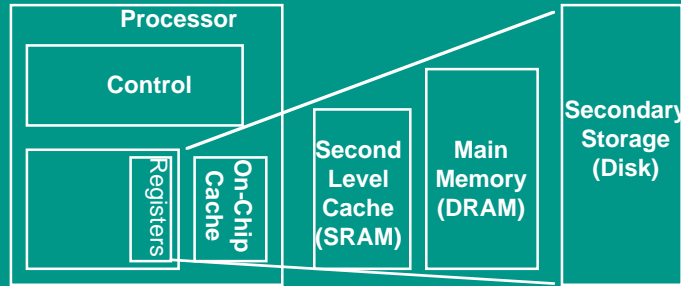
- **At any given time, data is copied between only 2 adjacent levels:**
 - **Upper Level:** the one closer to the processor
 - Smaller, faster, and uses more expensive technology
 - **Lower Level:** the one further away from the processor
 - Bigger, slower, and uses less expensive technology
- **Block:**
 - The minimum unit of information that can either be present or not present in the two level hierarchy



Memory, 8

Memory Hierarchy: Example

- By taking advantage of the principle of locality:
 - Provide as much memory as is available in cheapest technology.
 - Provide access at speed of fastest technology.

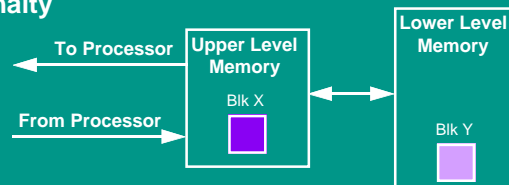


Name	Register	Cache	Main Memory	Disk memory
Speed	1ns	10ns	100ns	10,000,000ns (10s ms)
Size	100Bs	KBs	MBs	GBs

Memory, 9

Memory Hierarchy: Terminology

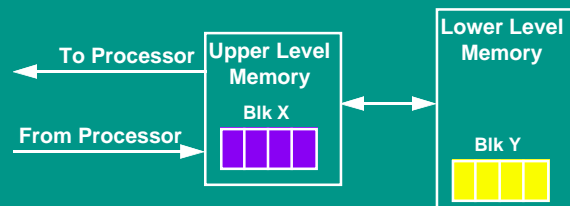
- Hit: data appears in a block in upper level (Block X)
 - Hit Rate: fraction of accesses found in upper level
 - Hit Time: Time to access the upper level consists of
 - RAM access time + Time to determine hit/miss
- Miss: get data from block in lower level (Block Y)
 - processor waits till data is fetched then restarts the access
 - Miss Rate = $1 - (\text{Hit Rate})$
 - Miss Penalty:
 - Time to get block from lower level
 - + time to replace in upper level
 - + time to deliver the block the processor
- Hit Time \ll Miss Penalty



Memory, 10

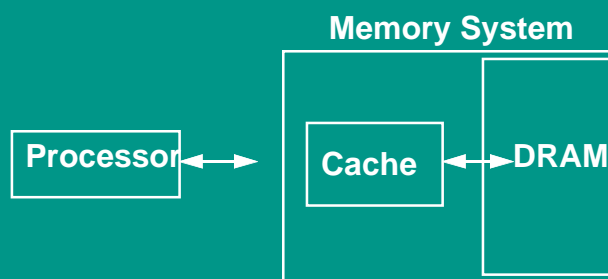
Memory Hierarchy: Exploiting Locality

- **Temporal Locality:** If an item is referenced, it will tend to be referenced again soon.
 - Keep more recently accessed data items closer to the processor
- **Spatial Locality:** If an item is referenced, items whose addresses are close by tend to be referenced soon.
 - Move blocks consisting of contiguous words to the upper levels



Memory. 11

Caches: the first transparent level



- **Motivation:**
 - Large memories (DRAM) slow; Small memories (SRAM) fast
- **Make the *average access time* small by:**
 - Servicing most accesses from small, fast memory.
- **Reduce *bandwidth* required of the large memory**

Memory. 12

Cache Access Example-1

Start Up

Access 000 01
miss

V	Tag	Data

Called a compulsory or cold start miss

Write Tag & Set Valid

V	Tag	Data
1	000	M [00001]

Miss Handling: Load Data

Memory, 15

Cache Access Example-2

Access 001 10
miss

V	Tag	Data
1	000	M [00001]

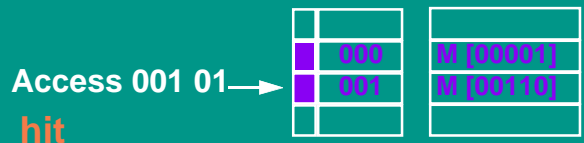
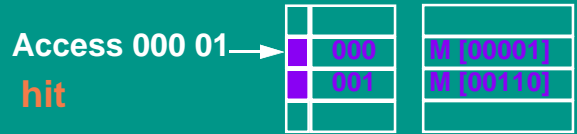
Write Tag & Set Valid

V	Tag	Data
1	000	M [00001]
1	001	M [00110]

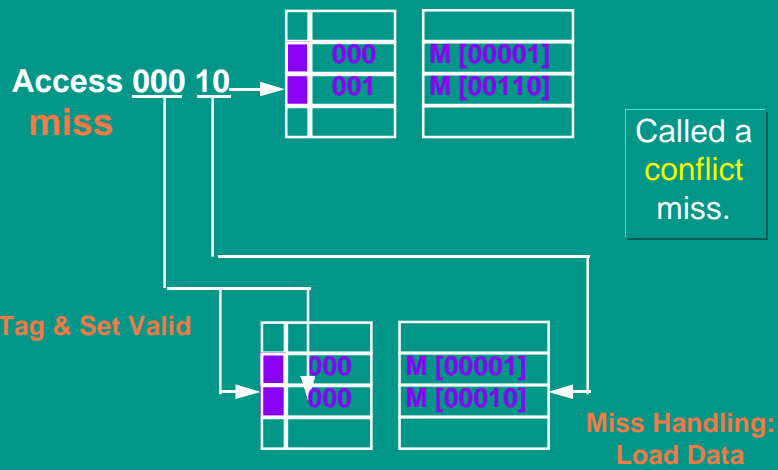
Miss Handling: Load Data

Memory, 16

Cache Access Example-3



Cache Access Example-4



Definition of a Cache Block

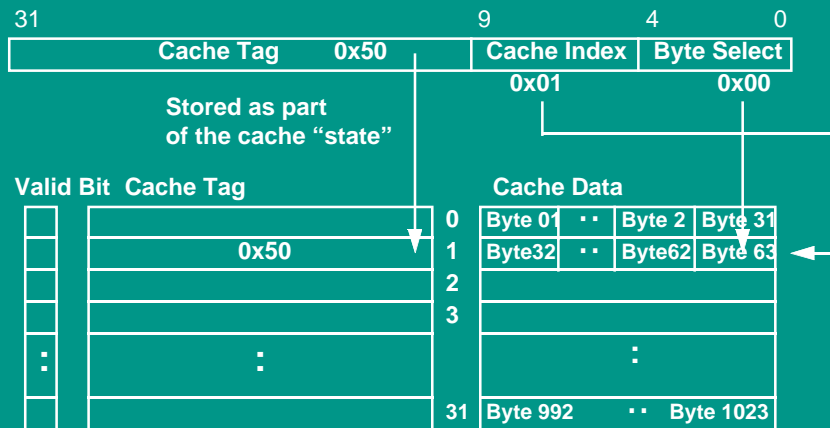
- **Cache Block**
 - a cache entry that has in its own cache tag
- **Our previous “extreme” example:**
 - 4-byte Direct Mapped cache: Block Size = 1 Byte
 - Takes advantage of Temporal Locality:
 - If a byte is referenced, it will tend to be referenced soon.
 - Did not take advantage of Spatial Locality:
 - If a byte is referenced, adjacent bytes will be referenced soon.
- **Take advantage of Spatial Locality:**
 - increase the block size



Memory 19

Example: 1 KB Cache, 32 B Blocks

- **For a 2^N byte cache:**
 - Uppermost (32 - N) bits are always the Cache Tag
 - Lowest M bits are the Byte Select (Block Size = 2^M)

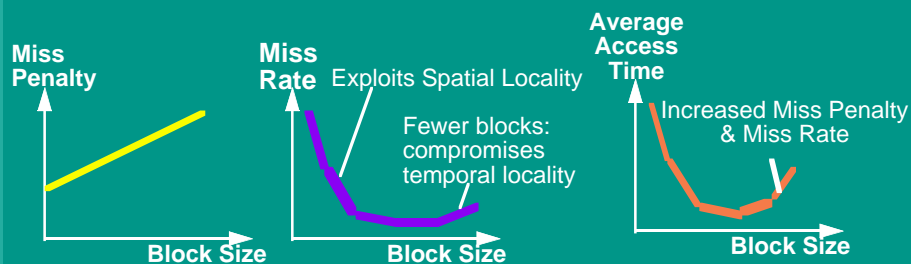


Memory 20

Block Size Tradeoff

- In general, larger block size take advantage of spatial locality *but*:
 - Larger block size means larger miss penalty:
 - Takes longer time to transfer the block
 - If block size is too big
 - average access time goes up
 - miss rate (MR) can even go up

$$\text{Average Access Time} = \text{Hit Time} \times (1 - \text{MR}) + \text{Miss Penalty} \times \text{MR}$$



Memory, 21

Another Extreme Example

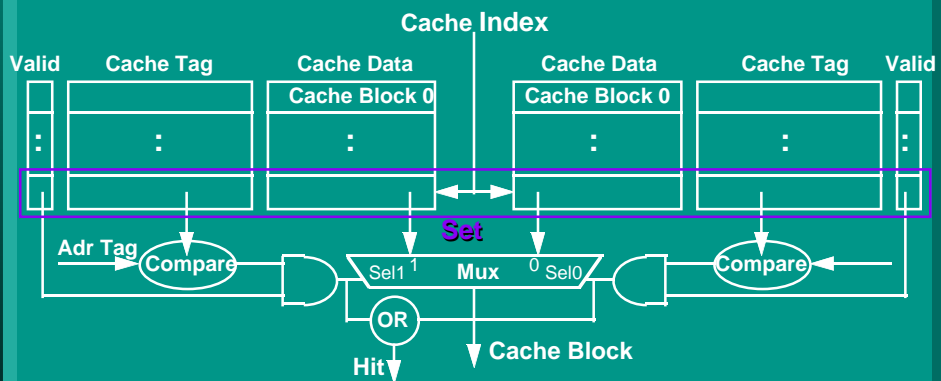
Valid Bit	Cache Tag	Cache Data			
		Byte 0	Byte 1	Byte 2	Byte 3
<input type="checkbox"/>		0			

- Cache Size = 4 bytes; Block Size = 4 bytes
 - Only *one* entry in the cache
- Caches assume: if an item is accessed, likely that it will be accessed again soon
 - But it is unlikely that it will be accessed again immediately!!!
 - The next access will likely to be a miss again
 - Continually loading data into the cache but discard (force out) them before they are used again
 - Worst nightmare of a cache designer: Ping Pong Effect
- These misses are misses caused by:
 - Different memory locations mapped to same cache index
 - Solution 1: make the cache size bigger
 - Solution 2: Multiple entries for the same Cache Index

Memory, 22

A Two-way Set Associative Cache

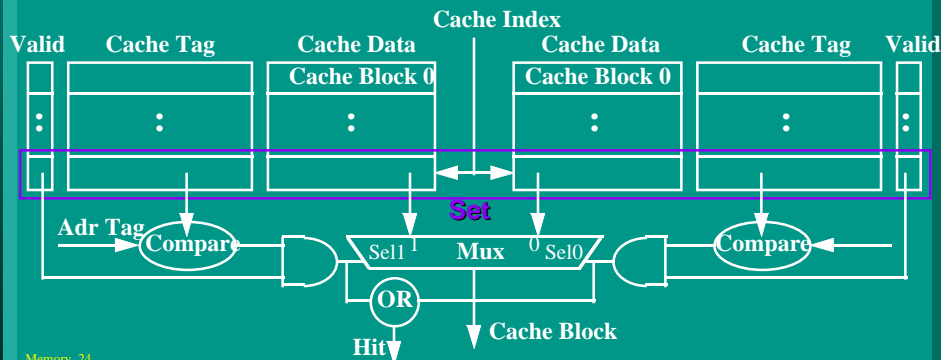
- N-way set associative:
 - N entries per Cache Index
 - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
 - Cache Index selects a "set"
 - The two tags in the set are compared in parallel
 - Data is selected based on tag



Memory_23

Disadvantage of Set Associative Cache

- N-way Set Associative vs. Direct Mapped:
 - N comparators vs. 1
 - Extra mux delay for data
 - Data comes *after* Hit/Miss
- Direct mapped cache:
 - Data available *before* Hit/Miss
 - Assume hit & continue.
 - Recover later if miss.



Memory_24

Which entry to replace on a miss?

- **Direct Mapped Cache:**
 - Each memory location mapped to only 1 cache location
 - No need to make any decision!
 - New item replaces previous item in that cache location
- **N-way Set Associative Cache:**
 - Each memory location has a choice of N cache locations
- **Fully Associative Cache:**
 - Each memory location can be placed in *any* cache location
- **Cache miss in N-way Set Associative or Fully Associative Cache:**
 - Bring in new block from memory
 - Throw out a cache block to make room for the new block
 - Need to make a decision on which block to throw out!

Memory, 27

Cache Block Replacement Policy

- **Random Replacement:**
 - Hardware randomly selects a cache item and throws it out
 - actually “pseudo random”
 - typically selected by counter or pointer
- **Least Recently Used (LRU):**
 - Hardware keeps track of access history
 - Replaces entry that has not been used for the longest time
 - Tricky to implement as associativity grows
 - need counters for each entry in the set
- **Not much difference in practice**
 - especially as associativity grows

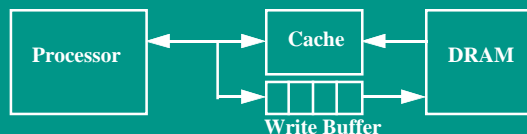
Memory, 28

Cache Write Policy

- Cache read much easier to handle than cache write
 - Read does not change value of data
- Cache write:
 - Need to keep data in the cache and memory consistent
- Two options:
 - Write Back: write to cache only.
 - Write the cache block to memory when that cache block is being replaced on a cache miss.
 - Reduces the memory bandwidth required
 - Keep a “dirty” bit for each cache block—set if block is written
 - Control can be complex
 - Write Through: write to cache and memory at the same time.
 - What!!! How can this be? Isn't memory too slow for this?

Memory_29

Write Buffer for Write Through



- Use Write Buffer between the Cache and Memory
 - Processor: writes data into the cache and the write buffer
 - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
 - Typically small number of entries (4-8)
 - Works fine if: Rate of stores < 1 / DRAM write cycle
 - As Rate of stores approaches 1 / DRAM write cycle
 - write buffer fills up
 - stall processor to allow memory to catch up
 - happens due to bursts in write rate even if average rate is OK

Memory_30

Cache Performance

- **CPU Time = (CPU Cycles + Memory Stall Cycles) x Clock cycle time**
- **Memory system affects:**
 - Memory stall cycles: cache miss stalls + write buffer stalls
 - Clock cycle time: since cache access often determines clock speed for a processor

Memory stall cycles = Read stall cycles + Write stall cycles
Read stall cycles = Read Miss Rate x #Reads x Read Miss Penalty
- **For write back cache**

Write stall cycles = Write Miss Rate x #Writes x Write Miss Penalty

 - combine read and write components:
Memory stall cycles = Miss Rate x #Accesses x Miss Penalty
Memory stall CPI = Miss Rate/instruction x Miss Penalty
- **For Write through caches: add write buffer stalls**

Memory. 31

Cache Performance: Example

- **Assume:**
 - Miss rate for instructions = 5%
 - Miss rate for data = 8%
 - Data references per instruction = 0.4
 - CPI with perfect cache = 1.4
 - Miss penalty = 20 cycles
- **Find performance relative to no misses (same CR)**

Misses/instruction = $0.05 + 0.08 \times 0.4 = 0.08$
Miss stall CPI = $0.08 \times 20 = 1.6$

 - Performance is ratio of CPIs:

$$\frac{\text{Performance no misses}}{\text{Performance w. misses}} = \frac{\text{CPI w. misses}}{\text{CPI no misses}} = \frac{1.4 + 1.6}{1.4} = 2.1$$

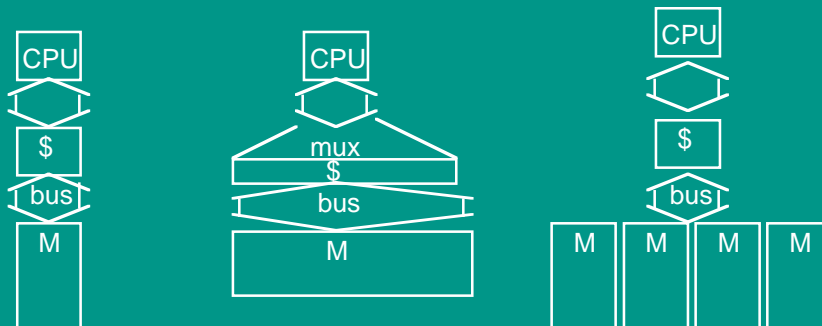
Memory. 32

Improving Cache Performance

- **Cache Performance is determined by:**
 - Average Memory Access Time = Hit Time + Miss Rate x Miss Penalty
- **Use better technology:**
 - faster RAMs—cost + availability are limitations
- **Decrease Hit Time**
 - make cache smaller: increases miss rate
 - use direct mapped: increases miss rate
- **Decrease Miss Rate**
 - Make cache larger: increases hit time
 - Add associativity: increase hit time
 - Increase block size: increases miss penalty
- **Decrease Miss Penalty**
 - Reduce transfer time component of miss penalty
 - Add another level of cache

Memory_33

Reducing Memory Transfer Time

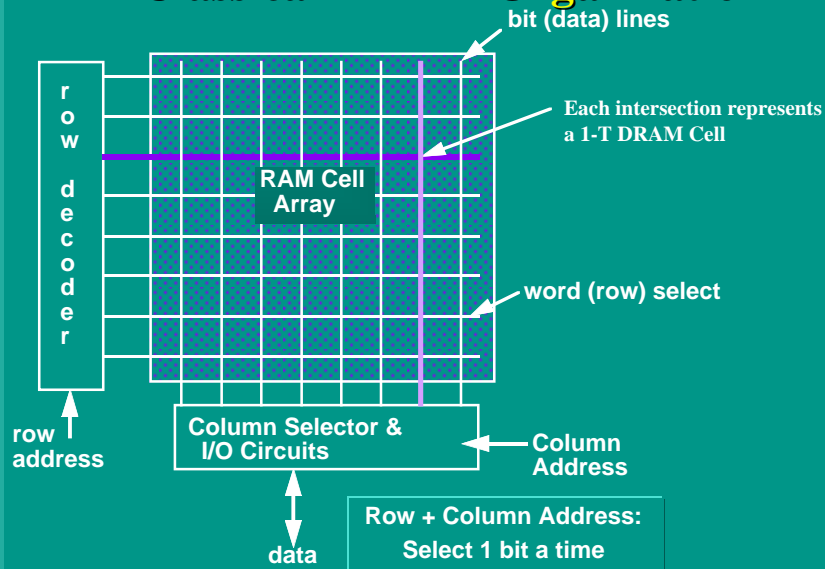


Solution 1 High BW DRAM
Solution 2 Wide Path Between Memory & Cache
Solution 3 Memory Interleaving

Examples:
 Page Mode DRAM
 SDRAM
 CDRAM
 RAMbus

Memory_34

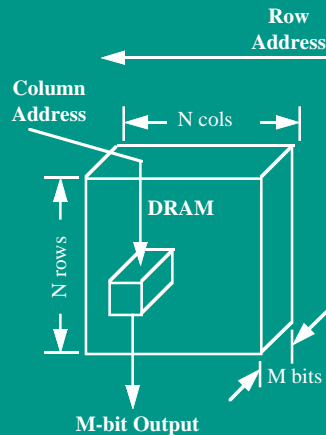
Classical DRAM Organization



Memory_35

Fast Page Mode DRAM

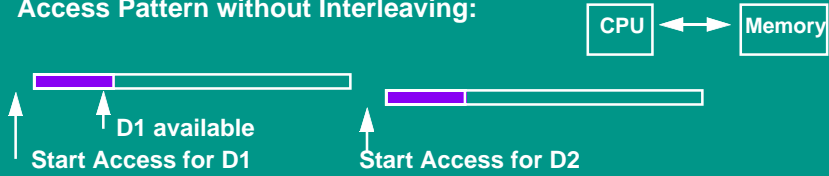
- Regular DRAM Organization:
 - N rows x N column x M-bit
 - Read & Write M-bit at a time
 - Each M-bit access requires RAS / CAS cycle
- Fast Page Mode DRAM
 - N x M "register" to save a row
 - only CAS is needed



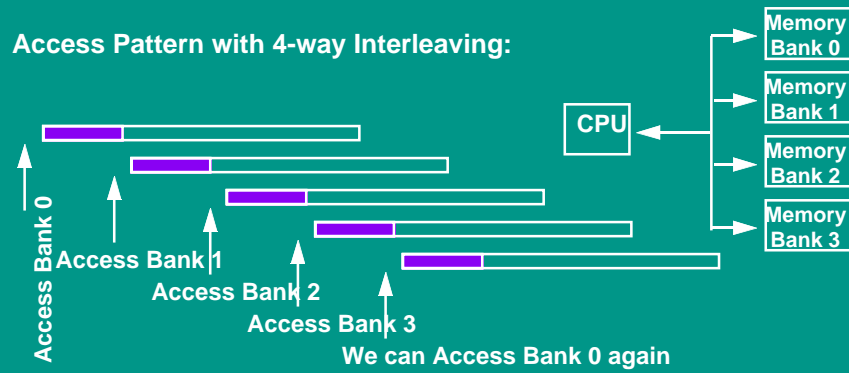
Memory_36

Increasing Bandwidth - Interleaving

Access Pattern without Interleaving:

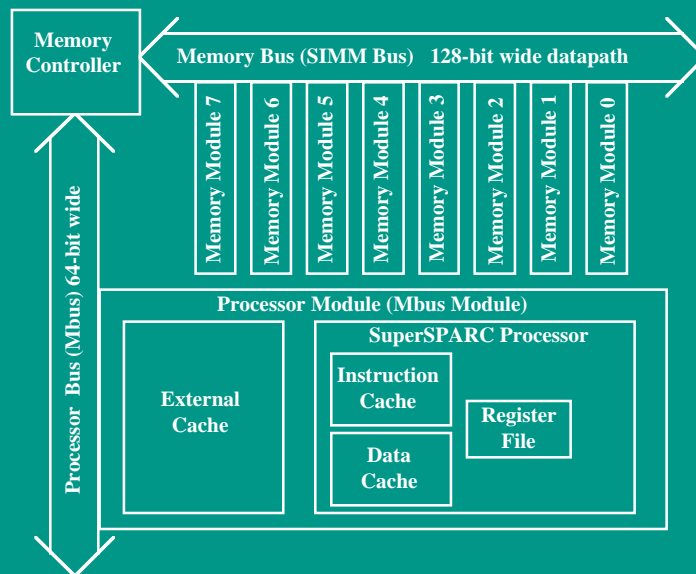


Access Pattern with 4-way Interleaving:



Memory_37

Example: SS 20's Memory System



Memory_38

Summary:

- Principle of Locality:
 - Temporal Locality: Locality in Time
 - Spatial Locality: Locality in Space
- Cost/performance tradeoff in memory technologies
- Three Major Categories of Cache Misses:
 - Compulsory Misses: first-time access
 - Conflict Misses: increase associativity or size
 - Capacity Misses: increase cache size
- Write Policy:
 - Write Through: need a write buffer.
 - Write Back: control can be complex

Memory_39

Virtual Memory

- Provides illusion of very large memory
 - Sum of memory of all job > physical memory
 - Address space of each job > physical memory
- Allows available (fast and expensive) physical memory to be well utilized
- Simplifies memory management (key motive today)
- Exploits hierarchy to reduce average access time
- Involves ≥ 2 storage levels: primary & secondary
- Virtual address: address used by programmer
- Virtual address space: collection of addressible locations (addresses)
- Memory (Physical) Address: address of word in physical memory

Memory_40

Basics of Virtual Memory (VM)

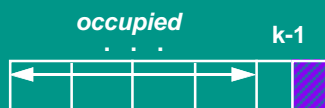
- Maps virtual addresses to physical addresses
 - called *address translation*
 - provides protection to allow sharing physical memory
- Misses handled by operating system
 - miss time very large: OS can do it efficiently
- Characteristics of blocks to transfer between secondary and main storage
 - variable size
 - fixed size
- Placement of blocks in memory
 - depends on fixed vs. variable
- When to fetch block
 - on demand
 - before needed

Memory. 41

Fixed vs. Variable Blocks

- Two major issues (there are others):
 - Easy of placing blocks: much easier for fixed blocks
 - *Fragmentation*: areas of memory unavailable for allocation
 - *External Fragmentation*: Space left between blocks.
 - problem for variable sized blocks: related to size of blocks
 - *Internal Fragmentation*:
 - program \neq integral # of fixed sized blocks
 - part of the last block is "wasted"—related to block size

Internal



External

Three different placements

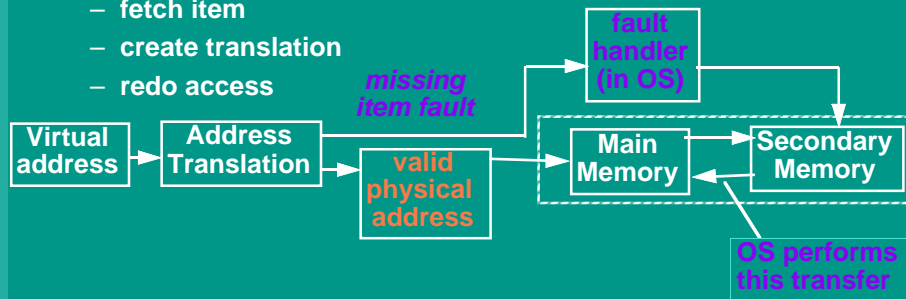


Would like to place this block

Memory. 42

Address Translation

- Program uses virtual addresses
 - provides relocation: a program can be loaded anywhere in memory—need not be compiled knowing where it will run
- Memory accessed with physical addresses
- No translation is a fault
 - fetch item
 - create translation
 - redo access



Memory_43

Exceptions

- Unusual condition causing transfer of control to OS
 - Examples:
 - integer overflow, translation fault, HW failure, I/O interrupt
- OS handles exception and may restart execution
 - Saves state of executing process, including address of cause of exception
 - Determine cause of exception
 - exception vector address: address where OS is initiated
 - register that records reasons (MIPS Cause register)
 - May need to restart exception as if nothing happened
 - Hardest type of exceptions
 - Page faults are like these—transparent to the user
 - Hardware may implement precise exceptions
 - “Hardware stops on a dime”—no extra instructions executed
 - SW may need to do extra work to handle these

Memory_44

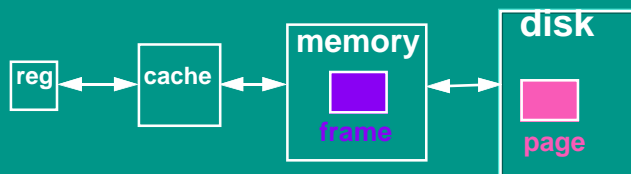
Implementing Protection

- Address translation mechanism allows sharing of memory by multiple users (and the OS) while maintaining protection from other users.
- Each process has its own address space
 - process = address space + processor state (registers + PC)
 - address space contains code + data
 - active when the process is running on CPU
 - processor state loaded (including PC), address space mapped
 - process state = everything that needs to be saved and restored to stop and restart process
 - simplifies compile & loading by using virtual address space
- How does this happen?
 - Each process has its own translation table.
 - Translation tables changed only by the OS.
 - Hence, a process cannot gain access to another process's address space or create unallocated pages.

Memory_45

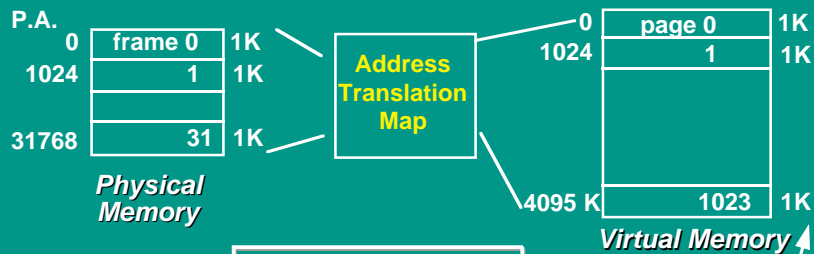
Paging: the Most Common VM

- Blocks:
 - virtual and physical address space partitioned into blocks of equal size
 - virtual address space blocks called *pages*
 - physical address space blocks called *frames* (page frames)
- Placement:
 - fully associative: any page in any frame
- Pages are fetched on demand



Memory_46

Paging Organization



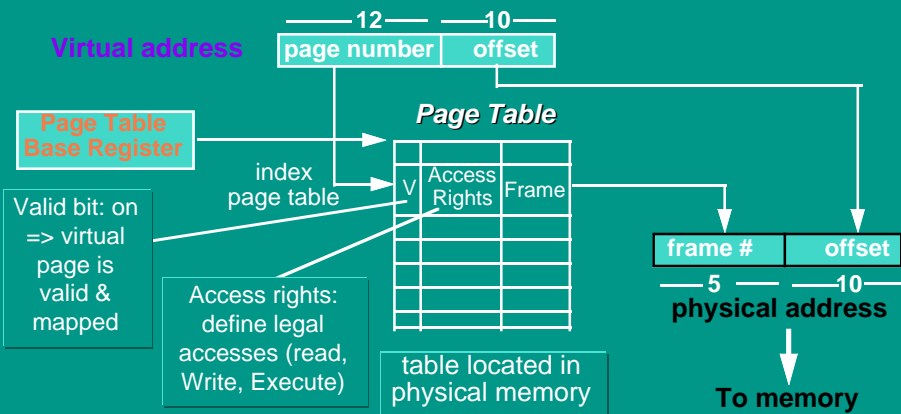
Since any virtual page can map to any frame, paging provides fully associative placement.

unit of mapping also unit of transfer from virtual to physical memory

Memory 47

Mapping Process

- Mapping is done by a page table—kept in memory
- Page size is power of two:
 - physical address replaces upper portion of virtual address
 - lower portion (page offset or displacement) is unchanged



Memory 48

Address Translation Algorithm

- If $V=1$ mapping legal
 - Check access rights (R, R/W, X) against access, if allowed
 - generate physical address and proceed
 - If access rights do not match, generate **protection fault**.
- If $V \neq 1$ page is not currently mapped
 - generate **page fault**
- Faults are exceptions and are handled by OS
 - page fault: fetch page, create map entry, restart process
 - another user process is run while the page is brought in by a context switch
 - protection fault: often program error, check protection and terminate program or change access

Memory, 49

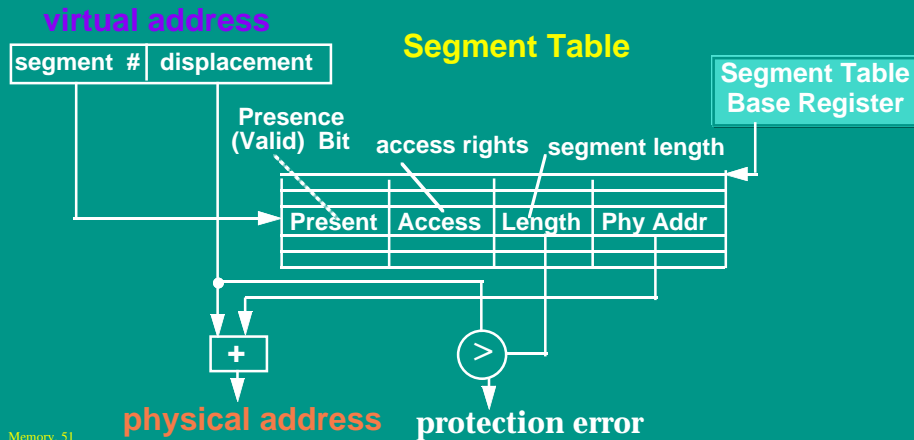
Page (Re)placement and Write Policy

- When a page fault occurs, choose a page to replace:
 - Fully associative so any frame/page is a candidate
 - Choose empty one if it exists.
 - Choose either (just as we did for cache):
 - LRU: approximated with Reference or Use bit in page table
 - true LRU is too hard, since we would need to track the time of use or every page frame
 - Commonly used clock algorithms: resets Use bits periodically
 - Random: approximate as we did for cache
- Write policy: Always write-back
 - Keep a dirty bit: set if written; when replaced write-back
 - Many systems use SW page cleaner to write-back pages to make selection of replacement easier

Memory, 50

Segmentation

- Virtual memory with variable sized *segments*
- Provides relocation as well as virtualization
- As an alternative to, or combined with paging



Memory, 51

Segmentation-Based Addressing

- Three major drawbacks:
 - if segment register is specified as separate part of address:
 - done to expand address space beyond word length
 - segmentation yields a nonlinear address space:
 - segments are visible to programmer, since must specify a segment number and segments are not adjacent
 - unless segment is large enough for all code or data creates programming difficulties or inefficiencies
 - Placing variable sized objects is hard
 - not simple like fixed-sized pages
 - External fragmentation—more serious especially if wide variation in block size
- Another kind of segmentation:
 - Fixed-size segments on top of pages (MSBs = segment #)
 - Goal: to conserve and manage page table space
 - Called “segments”, but different: none of above drawbacks

Memory, 52

Choosing a Page Size

- **Factors favoring large pages**
 - smaller page tables
 - more efficient for large programs and large memories
 - fewer page faults, more efficient disk transfer
- **Factors favoring smaller pages**
 - time to start-up small processes (only a few pages)
 - internal fragmentation (significant only for small programs)
- **General trend is towards large pages**
 - 1978: 512 B
 - 1984: 4 KB
 - 1990: 16 KB
 - 199x: 64 KB
- **Many recent machines support multiple page sizes to balance competing forces.**

Memory, 53

Making VM Fast—TLBs

- **If page table is kept in memory**
 - all memory accesses require two accesses:
 - one for page table entry and one to get the actual data.
- **Translation Lookaside Buffer (TLB)**
 - Make a cache of recently use page table translations
 - Look all accesses up in TLB
 - Hit in TLB yields get physical page number
 - Miss in TLB: get translation from the page table and reload
 - TLB usually smaller than cache (each entry maps a page):
 - more associativity possible
 - similar speed to cache access
 - contains all bits needed to translate address, implement VM
 - treat like normal cache: write back/write through of status

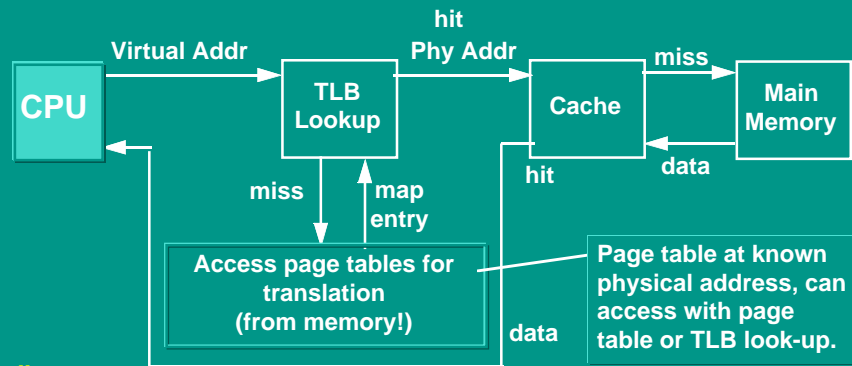
Typical TLB Entry

Virtual Address	Physical Address	Dirty	Reference	Valid	Access Rights
-----------------	------------------	-------	-----------	-------	---------------

Memory, 54

Memory access with a TLB

- Access TLB to get physical address
 - Miss: access memory to get translation
- Access cache to get data
 - Miss: access memory to get data
- Optimization: access TLB and cache in parallel



Memory, 55

Common Framework

- Common Policies in memory hierarchies
 - Block placement:
 - direct mapped
 - set associative
 - fully associative
 - Block location
 - Index with partial map (cache) or full map (page table)
 - Index + search (set associative)
 - Search (fully associative)
 - Replacement policy
 - random
 - LRU
 - Write policy
 - write-back
 - write-through

Memory, 56

Other Characteristics

- Key areas of difference are driven by performance
 - difference in access times/characteristics to levels
 - difference in sizes and costs of levels

<u>Hierarchy</u>	<u>Blocks</u>	<u>Size</u>	<u>Miss Handling</u>
Caches	Fixed	16-128 B	Hardware
Paging	Fixed	4KB-64KB	SW
Segments	Variable		SW
TLBs	Fixed	1-2 PTEs	SW or HW

Memory, 57

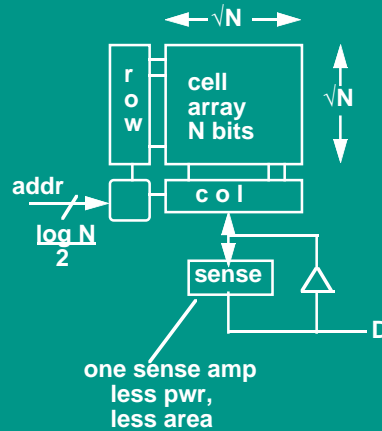
The Future

- Lots more stuff in memory hierarchies:
 - multilevel caches
 - subblocking
 - coherency
 - TLB design
 - virtual caches
 - write handling
- Memory Hierarchies will continue to be an area of great importance
 - gap between CPU demands and disk/DRAM capability
 - new innovative approaches needed to maintain pace and not be overwhelmed by slowest memory

Memory, 58

Introduction to DRAM

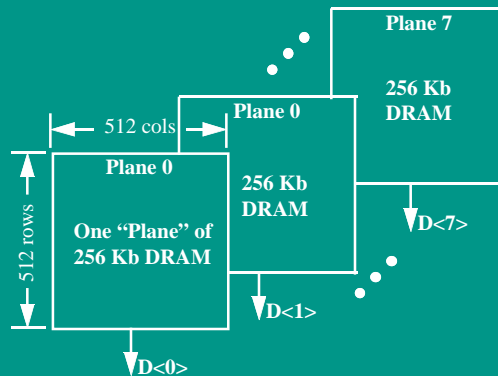
- Dynamic RAM (DRAM):
 - Refresh required
 - Very high density
 - Low power
 - Low cost per bit
 - # pins small:
 - Row address strobe (ras)
 - Col address strobe (cas)
 - Page mode operation



Memory_59

Typical DRAM Organization

- Typical DRAMs: access multiple bits in parallel
 - Example: 2 Mb = 256K x 8 = 512 rows x 512 cols x 8 bits
 - Row + column addresses applied to all planes in parallel



Memory_60

Cycle Time versus Access Time

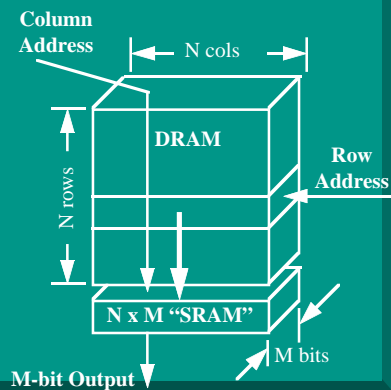


- DRAM Cycle Time \gg DRAM Access Time
- DRAM (Read/Write) Cycle Time :
 - How frequent can you initiate an access?
- DRAM (Read/Write) Access Time:
 - How quickly will you get what you want once you initiate an access?

Memory. 61

Fast Page Mode Operation

- Fast Page Mode DRAM
 - $N \times M$ “SRAM” to save a row
- After a row is read into the register
 - Only CAS is needed to access other M -bit blocks on that row
 - RAS_L remains asserted while CAS_L is toggled



Memory. 62

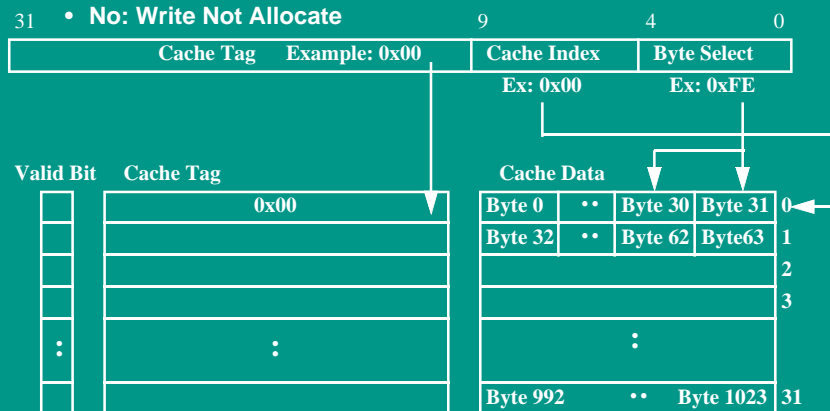
Summary:

- **Two Different Types of Locality:**
 - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
 - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
- **By taking advantage of the principle of locality:**
 - Present the user with as much memory as is available in the cheapest technology.
 - Provide access at the speed offered by the fastest technology.
- **DRAM is slow but cheap and dense:**
 - Good choice for presenting the user with a BIG memory system
- **SRAM is fast but expensive and not very dense:**
 - Good choice for providing the user FAST access time.

Memory, 63

Write Allocate versus Not Allocate

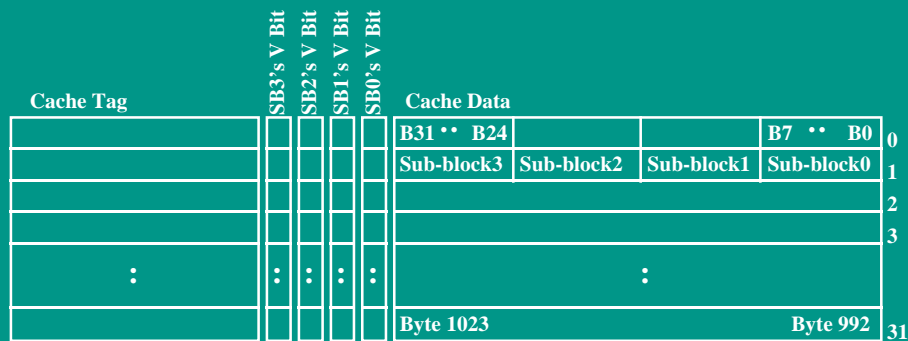
- **Assume: a 16-bit write to memory location 0x0 and causes a miss**
 - Do we read in the rest of the block (Byte 2, 3, ... 31)?
 - Yes: Write Allocate
 - No: Write Not Allocate



Memory, 64

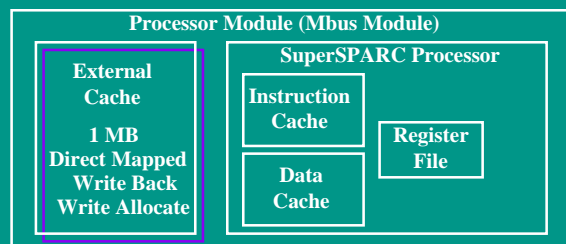
What is a Sub-block?

- **Sub-block:**
 - A unit within a block that has its own valid bit
 - E.g., 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block
 - Each cache entry will have: $32/8 = 4$ valid bits
- **Write miss: only bytes in the sub-block are brought in.**



Memory, 65

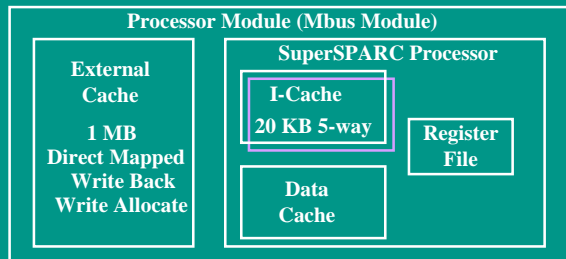
SPARCstation 20's External Cache



- **SPARCstation 20's External Cache:**
 - Size and organization: 1 MB, direct mapped
 - Block size: 128 B
 - Sub-block size: 32 B
 - Write Policy: Write back, write allocate

Memory, 66

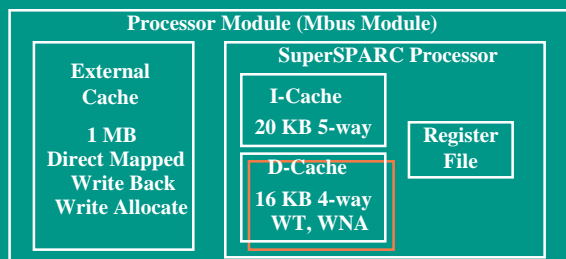
SS 20's Internal Instruction Cache



- SPARCstation 20's Internal Instruction Cache:
 - Size and organization: 20 KB, 5-way Set Associative
 - Block size: 64 B
 - Sub-block size: 32 B
 - Write Policy: Does not apply
- Note: Sub-block size the same as the External (L2) Cache

Memory, 67

SS 20's Internal Data Cache



- SPARCstation 20's Internal Data Cache:
 - Size and organization: 16 KB, 4-way Set Associative
 - Block size: 64 B
 - Sub-block size: 32 B
 - Write Policy: Write through, write not allocate
- Sub-block size the same as the External (L2) Cache

Memory, 68

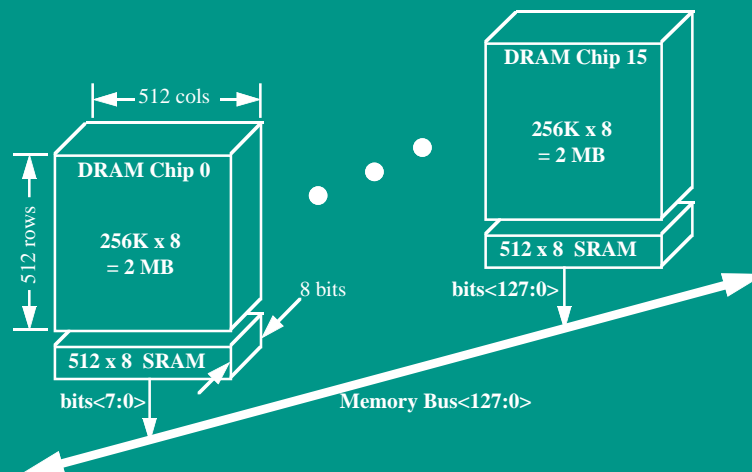
Two Interesting Questions?

- Why did they use N-way set associative cache internally?
 - Answer: A N-way set associative cache is like having N direct mapped caches in parallel. They want each of those N direct mapped cache to be 4 KB. Same as the “virtual page size.”
 - Virtual Page Size: cover in next week’s virtual memory lecture
- How many levels of cache does SPARCstation 20 has?
 - Answer: Three levels.
(1) Internal I & D caches, (2) External cache and (3) ...

Memory, 69

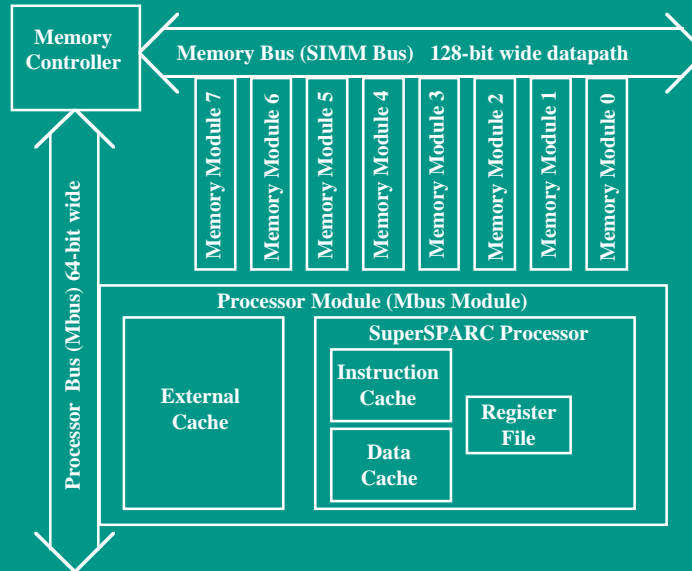
SS 20's Memory Module

- Supports a wide range of sizes:
 - Smallest 4 MB: 16 2Mb DRAM chips, 8 KB of Page Mode SRAM
 - Biggest: 64 MB: 32 16Mb chips, 16 KB of Page Mode SRAM



Memory, 70

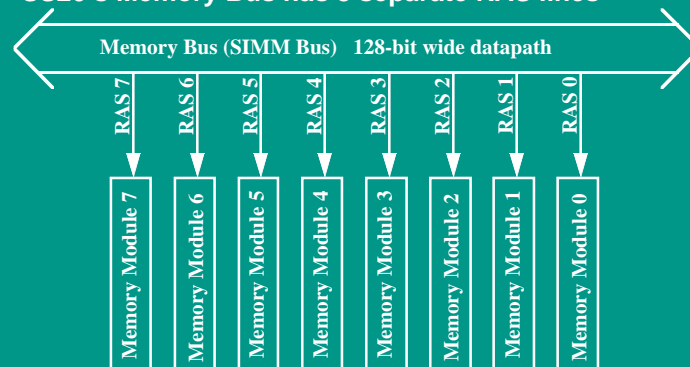
SS 20's Memory System Overview



Memory. 71

SS 20's Main Memory

- Up to 8 memory modules
- How do we select 1 out of the 8 memory modules?
Remember: every DRAM operation start with the assertion of RAS
 - SS20's Memory Bus has 8 separate RAS lines



Memory. 72

Page Table Fragmentation

- Page table size is proportional to size of address
- Large page tables if only a few pages are being used
- Easy case: all pages in use are contiguous
 - grow page table in one direction
 - keep a bound on table
- Harder cases:
 - use pages at opposite ends of address space
 - use pages sparsely
 - more significant with large (64-bit) addresses
 - Techniques to accommodate these:
 - Introduce two or more segments into address space
 - Use two or more page tables, each separately allocated and grown.

Memory, 73

Segmentation with Paging

- Simple case: two segments representing lower and upper half of address space
 - Page table for each segment with separate length register.
 - Each page table grows independently
 - Assumes pages are used contiguously from two ends of address space.
- More general scheme: many segments, two level page table with separate table for each segment active
 - efficient even if pages used sparsely
- In either case, number of page table entries determined by number of active pages.

Memory, 74